Problem 1 (15 points).

1a. (3 points). It looks at which address caused the fault (demand page selection).

1b. (3 points). It uses a page replacement algorithm, such as LRU, which hopefully picks a page frame containing a page that will not be used for a while.

1c. (3 points). If it has been changed since it was written on the disk, the latest version must be written to disk.

1d. (3 points). Both the maps for the old page (mark it invalid) and the new page (mark it valid), as well as the table that says which pages are in real memory.

1e. (3 points). Paging is infrequent if real memory is large enough to hold the working sets of all running processes.

Problem 2 (10 points).

2a. (5 points). $N = 1$, $Q[1] = \infty$.

2b. (5 points). $N = 1$, $Q[1] = T$.

Problem 3 (35 points). Solution in C++:

```
/*
 * We need two arrays indexed by station:  one for the boxes waiting
 * to be loaded on the train at a station and one for the boxes on the train
 * that should be unloaded at a particular station.  We also need to keep
 * track of the number of boxes on the train at any time.
 *
 * For each station, we need a mutual exclusion semaphore for accessing
 * the numWaiting arrays, a wait semaphore for loading, and a wait semaphore
 * for unloading.  We also need two semaphores for the train,
 * one for loaded boxes and one for unloaded boxes.
*/
```

```
    int numWaitingAtStation[N], numWaitingForStation[N], boxes;
    Semaphore mutex(1)[N];
    Semaphore waitingAtStation(0)[N], waitingForStation(0)[N];
    Semaphore load(0), unload(0);
    void train(int s) {
        int unloading, loading;
        mutex[s].P();
        unloading = numWaitingForStation[s];
        numWaitingForStation[s] = 0;
        boxes -= unloading;
        if (boxes + numWaitingAtStation[s] > B) {
          loading = B - boxes;
        } else {
          loading = numWaitingAtStation[s];
        }
        numWaitingAtStation[s] -= loading;
        mutex[s].V();
        boxes += loading;
        /* unload boxes destined for this station */
        for (int i = 0; i < unloading; i++) {
          waitingForStation[s].V();
          unload.P();
        }
        /* load boxes waiting at station */
        for (i = 0; i < loading; i++) }
          waitingAtStation[s].V();
          load.P();
        }
    }
    void sendbox(int fromStation, int toStation) {
        /* add to waiting at station */
        mutex[fromStation].P();
        ++numWaitingAtStation[fromStation];
        mutex[fromStation].V();
        /* wait for the train */
        waitingAtStation[fromStation].P();
        /* load */
        load.V();
        /* add to waiting for toStation */
        mutex[toStation].P();
        ++numWaitingForStation[toStation];
        mutex[toStation].V();
        /* wait for the train to reach toStation */
        waitingForStation[toStation].P();
        /* unload */
        unload.V();
    }
```